Flight Manager

Mário Branco – up202008219

Guilherme Coutinho – up202108872

Xavier Outeiro - up202108895

Classes

- Airline.cpp (airlines.csv)
- Flight.cpp (flights.csv)
- Airport.cpp (airports.csv)
- Graph.cpp (*.csv)

```cpp
//read airlines.csv
while (std::getline( &: in2,  &: token,  dlm: '\n')) {
    std::stringstream iss( s: token);
    std::vector<std::string> temp;
    std::string tempstr;
    currentAirline = new Airline();

    while ((std::getline( &: iss,  &: tempstr,  dlm: ',')))
        if (!tempstr.empty() && tempstr[tempstr.size() -
            tempstr.erase( pos: tempstr.size() - 1);
        temp.push_back(tempstr);
    }

    currentAirline->setCode(temp[0]);
    currentAirline->setName(temp[1]);
    currentAirline->setCallSign(temp[2]);
    currentAirline->setCountry(temp[3]);

    airlines[currentAirline->getCode()] = currentAirline;
}
```

```cpp
//reads airports.csv
while (std::getline( &: in1,  &: token,  dlm: '\n')) {
    std::stringstream iss( s: token);
    std::vector<std::string> temp;
    std::string tempstr;
    currentAirport = new Airport();


    while ((std::getline( &: iss,  &: tempstr,  dlm: ','))) {
        if (!tempstr.empty() && tempstr[tempstr.size() - 1] == '\r')
            tempstr.erase( pos: tempstr.size() - 1);
        temp.push_back(tempstr);

    }

    currentAirport->setCode(temp[0]);
    currentAirport->setName(temp[1]);
    currentAirport->setCity(temp[2]);
    currentAirport->setCountry(temp[3]);
    currentAirport->setLatitude(stod( str: temp[4]));
    currentAirport->setLongitude(stod( str: temp[5]));

    airports[currentAirport->getCode()] = currentAirport;
    n++;
}
```

```cpp
//reads flights.csv
while (std::getline( &: in3,  &: token,  dlm: '\n')) {
    std::stringstream iss( s: token);
    std::vector<std::string> temp;
    std::string tempstr;
    Airport *origin, *destination;
    Airline *airline;

    while ((std::getline( &: iss,  &: tempstr,  dlm: ',')
        if (!tempstr.empty() && tempstr[tempstr.size()
            tempstr.erase( pos: tempstr.size() - 1);
        temp.push_back(tempstr);
    }

    origin = airports[temp[0]];
    destination = airports[temp[1]];
    airline = airlines[temp[2]];
    origin->addFlight(destination, airline);
}
```

ReadFiles()

Leitura de Dados

De modo a realizar a leitura dos ficheiros .csv, utilizamos os métodos de iss (istringstream).

Decidimos guardar os dados lidos em vetores de strings temporárias, que posteriormente foram utilizados para a construção dos objetos (Airport, Airline, Flight). Após cada objeto ser criado, foram inseridos na sua estrutura de dados respetiva. (Aiport e Arline – Unordered Map na classe Graph) (Flight – vector<Flights> na classe aeroporto).

O grafo em si tem como nós os aeroportos e cada aerporto tem um vetor de Flights como edges. Dentro de cada Flight (edge), estão guardados parâmetros como a Airline utilizada, o aeroporto de destino, e a distância do voo).

Funcionalidades

- readFiles(): Lê ficheiros de input (.csv). O(n)
- calculateDistance(): Calcula a distância entre 2 aeroportos. O(1)
- bestTravel(): Retorna a melhor possibilidade de voos entre 2 aeroportos ou cidades. O(|V| + |E|)
- bestTravelAirport(): BestTravel() se forem escolhidos aeroportos. O(n)
- bestTravelCity(): BestTravel() se forem escolhidas cidades. O(n^2)

```
public:
    void readFiles(const std::string& file1, const std::string& file2, const std::string& file3);
    static double calculateDistance(Airport* a1, Airport* a2);
    Airport* bestTravel(Airport* origin, Airport* destination);
    void bestTravelAirport(Airport* origin, Airport* destination);
    void bestTravelCity(const string& origin, const string& destination);
    unordered_map<string,Airport*> getAirports();
    unordered_map<string,Airline*> getAirlines();
    int getNumberOfFlightsForAirport(const string &airportCode);
    int getNumberOfAirlinesAirport(const string &airportCode);
    void listAirlines(const string &airportCode);
    void listFlights(const string &airportCode);
    int getNumberOfReachableCities(const string &airportCode);
    int getNumberOfReachableCountries(const string &airportCode);
    Airport *airlineBestTravel(Airport *origin, Airport *destination, const string &airlineCode);
    void oneAirlineBestTravel(Airport *origin, Airport *destination, const string &airlineCode);
    Airport *multipleAirlineBestTravel(Airport *origin, Airport *destination, const vector<string> &airlineCodes);
    void multipleAirlinesPrint(Airport *origin, Airport *destination, const vector<string> &airlineCodes);
    int multipleFlightsReachableCities(const string &airportCode, int numFlights);
    int multipleFlightsReachableCountries(const string &airportCode, int numFlights);
};
```

Funcionalidades

- getNumberOfFlightsForAirport(): Retorna o número de voos que saem de um aeroporto em específico. O(1)
- listFlights(): Imprime os voos que saem desse aeroporto. O(n)
- getNumberOfAirlinesForAirport(): Retorna o número de companhias aéreas que têm voos num aeroporto específico. O(n)
- listAirlines(): Imprime os nomes das companhias aéreas com voos nesse aeroporto. O(n)
- getNumberOfReachableCities(): Retorna o número de cidades que são possíveis de atingir com apenas um voo partindo de um Aeroporto específico. O(n)

```
public:
    void readFiles(const std::string& file1, const std::string& file2, const std::string& file3);
    static double calculateDistance(Airport* a1, Airport* a2);
    Airport* bestTravel(Airport* origin, Airport* destination);
    void bestTravelAirport(Airport* origin, Airport* destination);
    void bestTravelCity(const string& origin, const string& destination);
    unordered_map<string,Airport*> getAirports();
    unordered_map<string,Airline*> getAirlines();
    int getNumberOfFlightsForAirport(const string &airportCode);
    int getNumberOfAirlinesAirport(const string &airportCode);
    void listAirlines(const string &airportCode);
    void listFlights(const string &airportCode);
    int getNumberOfReachableCities(const string &airportCode);
    int getNumberOfReachableCountries(const string &airportCode);
    Airport *airlineBestTravel(Airport *origin, Airport *destination, const string &airlineCode);
    void oneAirlineBestTravel(Airport *origin, Airport *destination, const string &airlineCode);
    Airport *multipleAirlineBestTravel(Airport *origin, Airport *destination, const vector<string> &airlineCodes);
    void multipleAirlinesPrint(Airport *origin, Airport *destination, const vector<string> &airlineCodes);
    int multipleFlightsReachableCities(const string &airportCode, int numFlights);
    int multipleFlightsReachableCountries(const string &airportCode, int numFlights);
};
```

Funcionalidades

- getNumberOfReachableCountries(): Retorna o número de países possíveis de atingir com apenas um voo partindo de um aeroporto em específico. O(n)
- multipleFlightsReachableCities(): Retorna o número de cidades que são possíveis de atingir com X voos partindo de um Aeroporto específico. (sendo X definido pelo utilizador) O(n^2)
- multipleFlightsReachableCountries(): Retorna o número de países possíveis de atingir com X voos partindo de um aeroporto em específico. (sendo X definido pelo utilizador) O(n^2)
- airlineBestTravel(): Retorna a melhor combinação de voos entre dois aeroportos usando apenas uma companhia aérea. O(|V| + |E|)

```
public:
    void readFiles(const std::string& file1, const std::string& file2, const std::string& file3);
    static double calculateDistance(Airport* a1, Airport* a2);
    Airport* bestTravel(Airport* origin, Airport* destination);
    void bestTravelAirport(Airport* origin, Airport* destination);
    void bestTravelCity(const string& origin, const string& destination);
    unordered_map<string,Airport*> getAirports();
    unordered_map<string,Airline*> getAirlines();
    int getNumberOfFlightsForAirport(const string &airportCode);
    int getNumberOfAirlinesAirport(const string &airportCode);
    void listAirlines(const string &airportCode);
    void listFlights(const string &airportCode);
    int getNumberOfReachableCities(const string &airportCode);
    int getNumberOfReachableCountries(const string &airportCode);
    Airport *airlineBestTravel(Airport *origin, Airport *destination, const string &airlineCode);
    void oneAirlineBestTravel(Airport *origin, Airport *destination, const string &airlineCode);
    Airport *multipleAirlineBestTravel(Airport *origin, Airport *destination, const vector<string> &airlineCodes);
    void multipleAirlinesPrint(Airport *origin, Airport *destination, const vector<string> &airlineCodes);
    int multipleFlightsReachableCities(const string &airportCode, int numFlights);
    int multipleFlightsReachableCountries(const string &airportCode, int numFlights);
};
```

Funcionalidades

- oneAirlineBestTravel(): Imprime o resultado da função airlineBestTravel(). O(n)
- multipleAirlinesBestTravel(): Calcula o melhor caminho possível entre dois aeroportos usando X companhias aéreas. (sendo X definido pelo utilizador) O(|V| + |E|)
- multipleAirlinesPrint(): Imprime o resultado da função multipleAirlinesBestTravel(). O(n)

```
public:
    void readFiles(const std::string& file1, const std::string& file2, const std::string& file3);
    static double calculateDistance(Airport* a1, Airport* a2);
    Airport* bestTravel(Airport* origin, Airport* destination);
    void bestTravelAirport(Airport* origin, Airport* destination);
    void bestTravelCity(const string& origin, const string& destination);
    unordered_map<string,Airport*> getAirports();
    unordered_map<string,Airline*> getAirlines();
    int getNumberOfFlightsForAirport(const string &airportCode);
    int getNumberOfAirlinesAirport(const string &airportCode);
    void listAirlines(const string &airportCode);
    void listFlights(const string &airportCode);
    int getNumberOfReachableCities(const string &airportCode);
    int getNumberOfReachableCountries(const string &airportCode);
    Airport *airlineBestTravel(Airport *origin, Airport *destination, const string &airlineCode);
    void oneAirlineBestTravel(Airport *origin, Airport *destination, const string &airlineCode);
    Airport *multipleAirlineBestTravel(Airport *origin, Airport *destination, const vector<string> &airlineCodes);
    void multipleAirlinesPrint(Airport *origin, Airport *destination, const vector<string> &airlineCodes);
    int multipleFlightsReachableCities(const string &airportCode, int numFlights);
    int multipleFlightsReachableCountries(const string &airportCode, int numFlights);
};
```

Interface (UI)

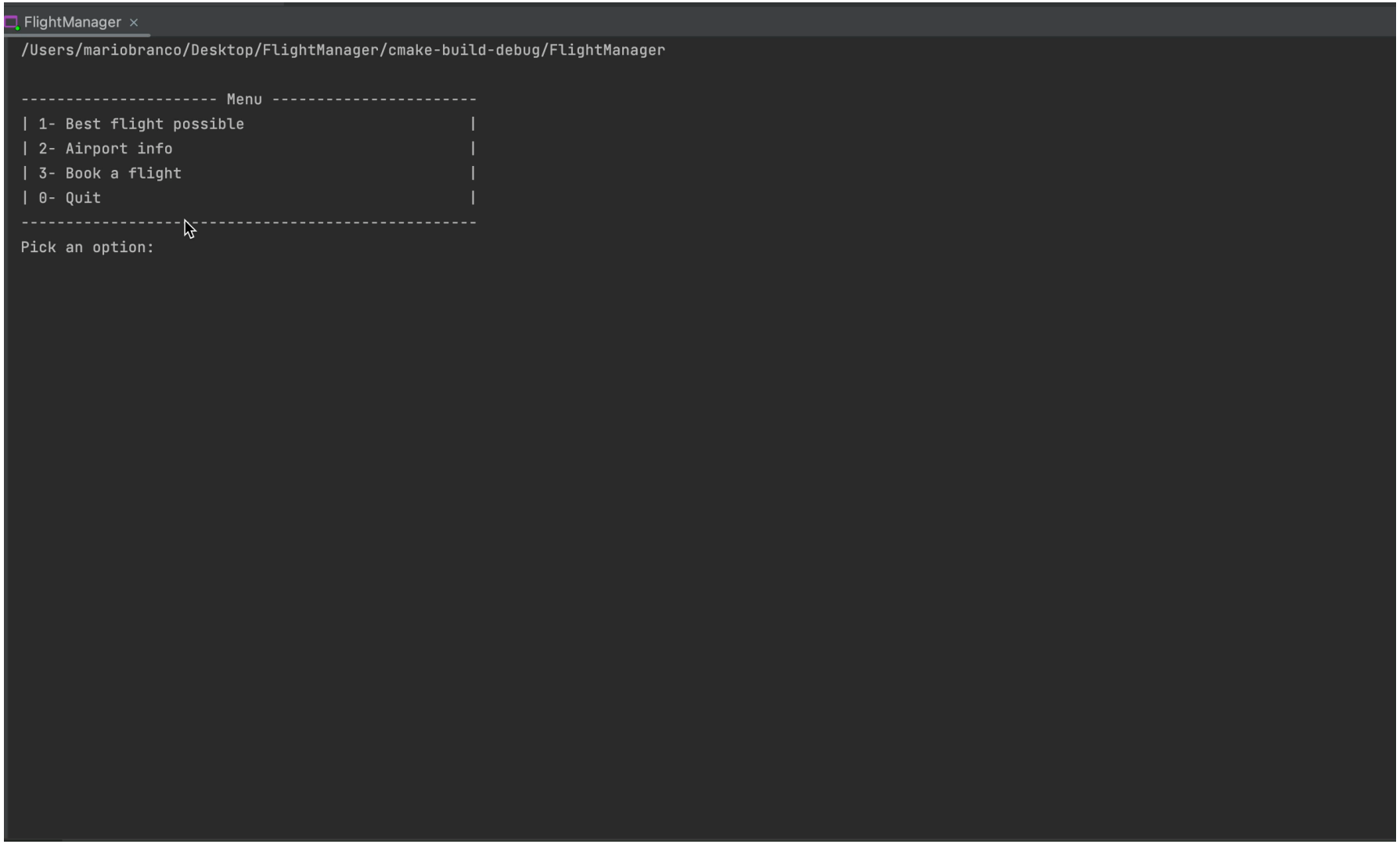
- Neste trabalho decidimos criar vários tipos de Menu's para as diferentes funcionalidades do programa. Melhorando assim a experiência do utilizador ao usar o programa.

```
    cout << "\n";
    cout << "---------------------- Menu ----------------------" << endl;
    cout << "| 1- List of Flights                               |" << endl;
    cout << "| 2- Number of Flights                             |" << endl;
    cout << "| 3- List of Airlines                              |" << endl;
    cout << "| 4- Number of Airlines                            |" << endl;
    cout << "| 5- Number of Reachable countries                 |" << endl;
    cout << "| 6- Number of Reachable cities                    |" << endl;
    cout << "| 0- Go back to menu                               |" << endl;
    cout << "---------------------------------------------------" << endl;
    cout << "Pick an option: ";
}

void pickAirline() {
    cout << "\n";
    cout << "---------------------------------------------------" << endl;
    cout << "| 1- Fly using one airline                         |" << endl;
    cout << "| 2- Fly using several airlines                    |" << endl;
    cout << "---------------------------------------------------" << endl;
    cout << "Pick an option: ";
}

void pickNumFlights() {
    cout << "\n";
    cout << "---------------------------------------------------" << endl;
    cout << "| 1- Find within one flight                        |" << endl;
```

```
void showMenu() {
    cout << "\n";
    cout << "---------------------- Menu ----------
    cout << "| 1- Best flight possible
    cout << "| 2- Airport info
    cout << "| 3- Book a flight
    cout << "| 0- Quit
    cout << "----------------------------------------
    cout << "Pick an option: ";
}

void showBestFlightMenu() {
    cout << "\n";
    cout << "----------------------------------------
    cout << "| 1- Search by airport
    cout << "| 2- Search by city
    cout << "----------------------------------------
    cout << "Pick an option: ";
}
```

```
case 3:
    cout << "============================" << endl;
    cout << "||                        ||" << endl;
    cout << "||  Book your flight!     ||" << endl;
    cout << "||                        ||" << endl;
    cout << "============================" << endl;
    cout << "\n";
    cout << "Select Origin Airport: ";
```

UX


FlightManager
/Users/mariobranco/Desktop/FlightManager/cmake-build-debug/FlightManager
---------- Menu ----------
| 1- Best flight possible |
| 2- Airport info |
| 3- Book a flight |
| 0- Quit |

Pick an option:

Melhor Funcionalidade

- Após realizar este trabalho somos capazes de destacar várias funcionalidades que achamos que estão bastante bem implementadas. Entre elas, destacamos a função bestTravel() que calcula o melhor caminho entre dois aeroportos usando BFS(Breadth-First-Search).

```
Airport* Graph::bestTravel(Airport *origin, Airport *destination) {

    for (auto it : pair<...> : airports) it.second->setVisited(false);
    for (auto it : pair<...> : airports) it.second->setScales({});
    queue<Airport *> q; // queue of unvisited nodes
    q.push( v: origin);
    origin->setVisited(true);
    origin->setDistance(0);
    while (!q.empty()) { // while there are still unvisited nodes
        Airport *u = q.front();
        q.pop();
        for (auto e : Flight : u->getFlights()) {
            Airport *w = e.getDestination();
            vector<pair<Airport*, Airline*>> current;
            pair<Airport*, Airline*> curr;
            if (!w->isVisited()) {
                q.push( v: w);
                w->setVisited(true);
                double dist = calculateDistance( a1: w,  a2: u);
                w->setDistance(u->getDistance() + dist);
                for (auto scale : pair<...> : u->getScales())
                    current.push_back(scale);
                curr.first = u;
                curr.second = e.getAirline();
                current.push_back(curr);
                w->setScales(current);
            }
            if (w == destination)
                return w;
        }
    }
    Airport *empty{};
    return empty;
```

Principais Dificuldades e Avaliação

- Interpretação do enunciado e perceber por onde começar a desenvolver o projeto.

- Esforço do grupo:
- Guilherme Coutinho – 100%
- Xavier Outeiro – 100%
- Mário Branco – 100%